Visions of Automation and Realities of Certification

Kelly J. Hayhurst* and C. Michael Holloway†
NASA Langley Research Center, Hampton, Virginia, 23681

Quite a lot of people envision automation as the solution to many of the problems in aviation and air transportation today, across all sectors: commercial, private, and military. This paper explains why some recent experiences with complex, highly-integrated, automated systems suggest that this vision will not be realized unless significant progress is made over the current state-of-the-practice in software system development and certification.

I. Introduction

THROUGHOUT the aerospace industry, we find blueprints, roadmaps, and other planning documents describing exciting new technologies envisioned to solve existing problems and facilitate our transition into the future National Airspace System (NAS). These documents play an important role in identifying high-level issues crucial to achieving the goals of safe and efficient operation within the NAS, and proposing how new technologies can be brought to bear on these issues. A common and consistent theme throughout these documents is solutions that rely heavily on highly-complex and automated systems—systems that are far more complex and pervasive than exist today. Although the word *software* actually appears infrequently in the plans, software is the enabling technology for these automated systems. In particular, proposed new technology solutions depend on software for both success and safety, and extend the complexity of today's aerospace systems beyond anything that has been successfully completed so far. This paper proposes that increasing awareness of the challenges associated with developing software-intensive systems is essential to realizing the visions.

The structure of the paper is as follows. Section II briefly enumerates some of the existing vision documents, and explains their unifying characteristic: reliance on automation as the primary means of solving problems. Section III explains the current reality of automated systems by describing several examples of recent failures of these systems. Section IV suggests a way forward from the current reality towards the ambitious future described in the vision documents. Finally, Section V contains brief concluding remarks.

II. A View of the Future

In the first few years of the 21st century, a number of documents have been developed describing visions for the future of aviation. Documents of this type have been developed by individual federal agencies such as the Federal Aviation Administration (FAA),[1,2] the National Aeronautics and Space Administration (NASA),[3] and the Department of Defense (DOD).[4] The Joint Planning and Development Office (JPDO), which has representatives from the DOD, Department of Commerce, Department of Homeland Security, White House Office of Science and Technology Policy, NASA, and FAA, has also developed a vision document.[5] These documents help scope and define the nature of the aerospace research that should be conducted in the coming years.

The various visions for the future of air travel have many common themes, such as:

1) schedule flexibility, with the possibility of on-demand travel service
2) a "hassle-free travel experience where safety and security measures, ticketing, and baggage checks are all transparent as the traveler or package moves easily through the airport and on and off the aircraft" [5].
3) increased sensitivity to environmental concerns, including reduced aircraft noise and emissions pollution
4) seamless integration of civil and military operations

Different documents recommend various technologies to achieve these themes. For example, to enable a "new era in flight," the NASA's Aeronautics Blueprint identifies four particular areas in which NASA will concentrate its aeronautics research: the airspace system, revolutionary vehicles, aviation security and safety, and a state-of-the-art

* Senior Research Scientist, Safety-Critical Avionics Systems Branch, 100 NASA Rd, MS 130, Hampton VA 23681.
† Senior Research Engineer, Floyd Thompson Fellow, Safety-Critical Avionics Systems Branch, 100 NASA Rd, MS 130, Hampton VA 23681.

educated workforce. Of the fifty-two specific technology solutions described in the Blueprint, our analysis showed that forty-one depended, at least in part, on software for success; and that for thirty-five of those forty-one, software was not only critical to success, but also to human safety.[6]

Cursory analysis of the other vision documents shows a similar pattern: significant increases in the use of automated systems. This envisioned future is a desirable one, with safer, cheaper, more dependable transportation available to more people than today. But how well connected is this future to the current reality? Examining the current state-of-the practice in developing and operating automated systems suggests an answer to this question.

III. Examples from the Current Reality

It is nearly impossible to make it through a day anywhere within the developed world without coming into contact with something containing software. Many aspects of modern society depend on software-intensive systems, and those system perform their intended function well. The commercial aviation industry has a noteworthy safety record with respect to software. "Since the early 1980s, technological advances have generated increasingly complex designs and systems for commercial aircraft. For example, advances have occurred in the use of software-based systems to monitor and control functions traditionally performed by cockpit crews… In many cases, the software-based systems have virtually replaced the hydraulic and mechanical systems used on earlier generations of aircraft." [7] Although software systems and related aspects have been recognized as contributing to a few commercial aviation accidents,[8-10] no commercial aircraft accident has yet been directly attributed to software.

Despite this commendable track record for software, evidence suggests that growing complexity may be straining our capacity to field safe systems. A recent *Aviation Week and Space Technology* article expresses the frustration with software's dual role as both a facilitator to success and a challenge to it: "Software is both the friend and foe of aerospace. It provides enormous flexibility, but ensnares with a complexity that can lead to programmatic and physical disaster. Engineers are slowly getting better at handling software as projects grow ever more sophisticated, but it still continues to bite."[11] In hopes of finding a way to prevent or mitigate those problems in the future, it is helpful to examine the problems with recent large-scale automation projects.

In the remainder of this section we examine software problems from FAA, NASA, and DoD projects. The Standard Terminal Automation Replacement System (STARS) exemplifies difficulties with cost and schedule; the Mars Climate Orbiter and Mars Polar Lander represent infamous software failures; and the F/A-22 stealth fighter exemplifies significant software integration woes. These are just a few of the projects from the aerospace industry that have suffered software problems. If one looks at each project individually, the software problems may not necessarily seem remarkable. When the projects are examined together, the commonality of the problems is striking. In the discussions below, focus is specifically directed to software aspects of the project that went awry, despite other aspects of the projects that are more positive, in order to fully appreciate the complexity of software issues and identify areas where additional research may be helpful to minimize future project risks due to software. For each project, we examine the original project plans, prominent software problems, and the impact of the software problems on the project.

A. STARS

Since September 1996, the FAA has been developing STARS to replace outdated computer equipment that air traffic controllers currently use to control air space within 5 to 50 nautical miles of an airport. The new air traffic controller workstations are designed to have new color displays, processors, and software that allow for easy and rapid incorporation of new air traffic management tools. The original plan called for installation of STARS in 172 facilities at an estimated cost of $940 million, with implementation to begin in 1998 and be completed by 2005. The original resource estimate was based on using commercial-off-the-shelf technology with limited development of new software. Early in the program, however, the FAA decided to turn STARS into a more customized system. This decision was based on feedback about computer-human interface issues from FAA controllers and maintenance technicians who were using preliminary versions of STARS. The decision resulted in more extensive new software development than originally planned, and significant cost and schedule extensions.[12]

The rework required for STARS caused a four-year delay in initial installations. Despite the rework to correct computer-human interface issues, those installations were not without problems. The first deployment of STARS, in an early display configuration, was accomplished in El Paso, Texas, in April 2002. Two months after the El Paso deployment, STARS was deployed in Syracuse, New York. Since those deployments, controllers reported numerous emergency and high priority issues. For example, not all aircraft that should have been automatically displayed on the controllers' screens by STARS were being displayed properly.[13, 14]

While the full STARS configuration has been operational since 2002 at several major airports, including Philadelphia and Detroit, problems continue to be reported, including the failure to show some departing airplanes, duplicating others, and falsely identifying some objects.[15] The problems with STARS have been attributed to a number of software issues including:

1) Substantial requirements changes, due at least in part to over reliance on commercial-off-the-shelf software that was intended to save money
2) Underestimating the cost of site adaptation
3) Failure to adequately address computer-human interface issues
4) Underestimating the value of testing (deploying before adequate testing was complete)
5) Underestimating the significance of trouble reports

Since April 2004, the FAA has re-baselined the STARS program, with a phase one deployment to 50 sites for $1.46 billion, and a phase two deployment plan for 162 sites at an expected cost of $2.76 billion with a targeted completion time of 2012.[16]

B. Mars Climate Orbiter, Mars Polar Lander

NASA has also experienced its share of software problems, with results that are fairly well known. Two of these publicly recognized failures are the losses in 1999 of the Mars Climate Orbiter (MCO) and the Mars Polar Lander (MPL), along with two Deep Space 2 probes. MCO and MPL were the second and third projects in a series of robotic missions to explore Mars as part of NASA's Mars Surveyor program. MCO was intended to orbit Mars as a weather satellite and provide a communications relay for the MPL; and the MPL was designed to study volatiles and climate history on the surface of Mars. The Deep Space probes, which were launched with the MPL, were designed to collect and analyze soil samples from the planet surface. The project costs totaled $327.6 million for both orbiter and lander, not including the Deep Space 2 probes: $193.1 million for spacecraft development, $91.7 million for launch, and $42.8 million for mission operations.

Mars Climate Orbiter was launched December 11, 1998, and the last signal was received from it on September 23, 1999 following Mars orbit insertion. The MCO spacecraft was assumed lost. Mars Polar Lander was launched shortly after the launch of the Climate Orbiter on January 3, 1999, and arrived at Mars eleven months later. MPL lost contact with mission control approximately ten minutes before its expected landing, and was assumed crashed into the Martian surface.

Mishap investigation boards were convened to examine each failure and determine possible causes. According to the mishap investigation board for MCO, the root cause of the loss was "the failure to use metric units in the coding of a ground software file, 'Small Forces', used in trajectory models."[17] As a result, errors were introduced in the orbiter's trajectory estimate during its flight, leading to a significantly lower trajectory at the time of Mars insertion than expected, and the likely destruction of the orbiter in the Martian atmosphere. The MPL investigation team determined that the most likely cause of the loss of the lander was "premature shutdown of the descent engines, resulting from a vulnerability of the software to transient signals."[18]

At first glance, these causes may seem like fairly simple software problems. However, as with most accidents, there were many factors that contributed to the existence of these problems.

In the Climate Orbiter investigation, the following observations were made relevant to software. The software that implemented the Angular Momentum Desaturation (AMD) procedure on the spacecraft assumed metric units for the computation; this was consistent with the software specification. The corresponding ground software, however, assumed English units. This mismatch in units caused an underestimation of the effect of thruster firings, which led to the departure from the planned trajectory.

This seems like a particularly silly error. However, it is important to note that few of the software engineers realized that the rockets were fired as part of the AMD events. Also, the AMD software was called on far more often that was originally intended, in part because of novel design elements incorporated in the orbiter to reduce hardware costs. The novel design caused an increase in the complexity of associated software.[19]

In addition to the causal finding, the board investigating the Mars Climate Orbiter failure made the following determinations that are relevant to software:[17]

1) "[t]here was inadequate training of the MCO team on the importance of an acceptable approach to end to end testing of the small forces ground software."
2) "[e]nd-to-end testing to validate the small forces ground software performance and its applicability to the specification did not appear to be accomplished. It was not clear that the ground software independent verification and validation was accomplished for MCO. The interface control process and the verification of specific ground system interfaces was not completed or was completed with insufficient rigor."

3) "a number of reviews took place without the proper representation of key personnel," including operations navigation personnel.

The board investigating the Mars Polar Lander failure also made several observations concerning the development and testing of the software for the spacecraft:[18]

1) "full evaluation of system interaction between propulsion, thermal, and control was incomplete. Fault-tree analysis was treated inconsistently. The thermal and software system design activities lagged behind the design of other subsystems requiring these inputs. In some cases, consideration of potential failure modes was not adequately assessed."
2) "The flight software was not subjected to complete fault-insertion testing. Problems with post-landing fault-response algorithms … were uncovered in the course of the investigation. The touchdown sensing software was not tested with the lander in the flight configuration. Because of this, the software error was not discovered during the verification and validation program."
3) "The complexity of the interactions between the feed system, the thrusters, the structure, the G&C [Guidance and Control] sensors, and the G&C algorithms that the PWM approach creates, practically dictate that the only way of verifying the system with high confidence is with a full-scale closed-loop test of the system. This was prohibitive from a cost and schedule point of view and it was not done."

This third observation pertains to the choice to use pulse-width modulation (PWM) for controlling the thrust of the descent engines instead of a conventional throttle based system. This choice reduced the cost of the lander's hardware, but greatly increased the complexity of the software, because it had to calculate the exact duration of each engine pulse during the descent.[19]

Together, the findings from the two investigations indicate the following problems: (1) added software complexity due to design decisions intended to save money, (2) failure to involve the appropriate domain experts throughout system development, and (3) underestimating the importance of software testing.

C. F/A-22

Since 1986, the Air Force has been developing the F/A-22[‡] aircraft to replace its fleet of F-15 air superiority fighters. Attributes of the F/A-22 include the ability to be less detectable to adversaries, high speed capability for long ranges, and improved situational awareness for the pilot through integrated avionics. The Air Force originally planned to complete development of the F/A-22 in 1995, achieve initial operational capability by 1996, and ultimately purchase 750 aircraft. That plan has significantly changed. As of April 2005, completion of system development had been delayed by 10 years (it is expected this year), with a procurement estimate of only 178 aircraft.[20]

Cost increases and schedule delays over the 19 year development period have been attributed to numerous management and technical challenges. Software has been only one of those challenges, but a truly significant one. Dornheim writes "The slow development pace of the Air Force's F/A-22 fighter is partly due to software problems. The features and sophistication used to sell it have a dark side—large, complex software can be difficult to test thoroughly."[11]

In particular, software instability has been repeatedly cited as an on-going problem throughout development. In 2003, *Aviation Week & Space Technology* reported the following:[21]

> With the threat of continued software problems derailing U.S. Air Force plans for the F/A-22 stealth fighter, prime contractor Lockheed Martin has devised a way to gradually overcome long-standing glitches that have led to repeated crashes of onboard computers. Program officials report recent improvements in software stability, but the number of error-induced avionics shutdowns is still far above where it should be. … Dealing with the software's shortcomings is becoming increasingly important for program managers, with the stakes having increased significantly in recent months. The situation no longer represents only a technical obstacle, but has grown into a major political stumbling block. The Senate Armed Services Committee, which instigated a proposal to trim F/A-22 production next fiscal year by two aircraft, said that "the greatest challenge in the F/A-22 development program is one of software integration, which has resulted in a software instability problem that affected both the startup of the integrated weapons systems and the continuity of the system while in operation."

Also, in 2003 the Government Accountability Office (GAO)[§] found that the F/A-22 avionics had reportedly failed or shut down on numerous tests due to software problems. Specifically, the shutdowns occurred when the pilot attempted to use the radar, communication, navigation, identification, and electronic warfare system concurrently, something military pilots would generally expect their aircraft to be able to handle. According to a

[‡] The original designation for the aircraft was simply F-22. The Air Force changed the name to F/A-22 in 2002 to acknowledge the expanded air-to-ground attack capabilities not originally planned for the aircraft.

[§] The name of this organization was changed from General Accounting Office on July 7, 2004.

special team convened by the Air Force to address the F/A-22's software problems, "the unpredictable nature of the shutdowns was not surprising considering the complexity of the avionics system."[22]

Despite claims that the software instability problems had been solved,[23] an F/A-22 crashed 11 seconds after takeoff during a training mission at Nellis Air Force Base, Nevada, on December 20, 2004. As the accident report stated, "Immediately upon leaving the ground, the [mishap aircraft (MA)] began a series of un-commanded and progressively more violent yaw, roll, and pitch transients. Unable to control the aircraft, the [pilot] ejected seconds before the MA impacted the ground. The Accident Investigation Board President determined the cause of the mishap, supported by clear and convincing evidence, was an inoperative Flight Control System, resulting from a power interruption, which made the MA uncontrollable."[24]

According to the report, the pilot inadvertently triggered the failure of the rate sensor assembly (RSA) during a pre-flight maintenance check. The pilot shut down the engines during the pre-flight check, which caused the flight control system to momentarily lose power. "This interruption in the power supply then is linked to a known quirk in the RSA unit, which is programmed so that it could interpret a momentary power loss as an instruction to enter test mode, which freezes or 'latches' the [RSA] unit."[25] As late as August 2005, the integrated avionics suite was reportedly being redesigned "under a new upgrade programme intended to improve overall stability and add room for a new long-range datalink system."[26]

According to Philip Coyle, former chief weapons tester for the Pentagon, the F/A-22's avionics computers caused most of the program delays. "[C]omputers and software are the major pacing items for every big defense system. … Mastery of the computers and software that run weapon systems determines the pace of development and whether a program is within budget and on schedule. … The more complicated a system gets, it's more difficult to know what's in there."[27]

D. Summary of the Current Reality

A number of common software-related problems are evident among these projects, most notably in the areas of requirements, verification, and integration. In all of the examples, problems with software requirements are present: (1) problems with adequate requirements capture (knowing that the right system is completely and correctly specified at the start), (2) problems with changing requirements, and (3) problems with correct specification and implementation of requirements. The requirements problems were particularly obvious in STARS, where the extent of requirements changes in transitioning from a COTS-based system to a custom software system was a significant factor in the cost and schedule delays. In the Mars Climate Orbiter, the requirements problems were more subtle. In that case, incomplete, and consequently incorrect, understanding of how often the software would be used in the angular momentum desaturation events contributed to mission failure.

Insufficient verification, especially inadequate testing before deployment, was also a consistent theme in all of the examples. Software complexity compounded the difficulties with testing in all cases as well. Integration problems in many ways reflect a combination of requirements and verification issues. Successful integration requires a thorough knowledge of how all components work together and means for testing all of the interactions. The F/A-22 has clearly suffered significantly from these problems.

No one would dispute that the current reality on most large projects includes software problems, similar to those in our examples. But, does the evidence indicate that these problems are merely "glitches"—lapses in human judgment that should have been detected and corrected with existing processes and tools? Or, does the evidence point to fundamental problems in software engineering of complex projects that could eventually thwart certification of future technologies? That is, are we reaching a barrier in our ability to efficiently develop and deploy large automated systems?

Evidence across a broad spectrum of current National Airspace System (NAS) modernization efforts shows that we are potentially approaching that barrier. The Office of the Inspector General and the Government Accountability Office track the progress of all major FAA acquisition projects, such as STARS, intended to modernize and add new capabilities to the NAS. As of May 2005, of sixteen major acquisition projects being tracked, eleven are over budget with cost growth greater than $5.6 billion; nine have experienced schedule delays ranging from 2-12 years; and two projects have been deferred.[16] Those statistics, in themselves, are not necessarily remarkable for large projects. The fact that software has been singled out as *the* primary reason for these troubles is remarkable. According to the GAO, "[s]oftware development—the most critical component of key FAA modernization programs—has been the Achilles' heel of FAA's efforts to deliver programs on time and within budget."[28] In their most recent audit of the major acquisitions, the GAO specifically cited the following as contributors to cost, schedule, and performance problems: (1) adding requirements and/or unplanned work, (2) not sufficiently involving all necessary personnel throughout system development, and (3) underestimating the complexity of software development.[29]

The evidence is not limited to the aviation industry. A recent report by the Center for National Software Studies states the following: "Significant advances have been made and continue to be made in software technology, tools and practices. That was the good news. The bad news is that in the same period of time, the growth in pervasiveness and complexity of software has significantly outpaced that progress." The report goes on to identify a critical gap in the "tools and technology to routinely develop error-free software and the current state-of-the-art."[30]

Similarly, a recent article in the *IEEE Spectrum* magazine contains a table titled, "Software Hall of Shame," in which 31 failed software development projects from 1992-2005 are listed. The list spans many industries and includes both projects that were cancelled before deployment and those that had significant problems once deployed.[31]

The evidence about the current reality is clear. The state-of-the-practice is as follows: Developing safety-critical, software-intensive systems is difficult and expensive, and we appear to be reaching (or in some cases have already exceeded) the limit of complexity for which such systems can be developed at all.

IV. A Way Forward

Given the current state-of-the-practice, we suggest three steps that may be taken towards meeting challenges to the future vision.[**]

A. Step One.

The first step on the way forward is to understand that it is wrong to think of automation as a problem-free solution. Towards that end software knowledge is essential. On the surface, this may seem like a trivial and unnecessary step. On the contrary, software knowledge is elusive, as shown in the repeated underestimation of software complexity and development efforts in the example projects. The ubiquitous nature of software gives the appearance of maturity in software disciplines. For many common uses of software, the appearance matches the reality. Software disciplines are sufficiently mature to produce many useful systems and products. But for systems and applications with the complexity and criticality of the sort required to fulfill the future vision, the appearance of maturity is simply an illusion; the reality is an increasing array of software and safety problems. Recognizing these problems, and understanding how to, and how not to, approach discovering solutions to these problems will be quite difficult.

Fred Brooks demonstrated nearly twenty years ago that searching for a single "magic" solution (a.k.a. silver bullet) for software problems is futile: no one particular approach will be able to solve all the problems of software engineering.[32] Fifteen years later, Edsger Dijkstra argued that software engineering challenges are yet to be met, although "there is a widespread belief that computing science as such has been all but completed…This widespread belief, however, is only correct if we identify the goals of computing science with what has been accomplished and forget those goals that we failed to reach, even if they are too important to be ignored. I would therefore like to posit that computing's central challenge, 'How not to make a mess of it,' has *not* been met."[33]

If a highly automated future is to occur, then we must recognize that problems such as those described in this paper are not merely isolated instances, attributable to shortcomings in particular individuals or organizations, but rather they are instantiations of fundamental gaps in our understanding of how to efficiently and effectively develop complex software systems.

B. Step Two.

The second step on the way forward is the creation, advocacy, and execution of a robust software research program that concentrates on eliminating these fundamental gaps, many of which were exposed in the projects described in Section III. In previous work, [6] we characterized the gaps as follows:

1) How to efficiently, accurately, and completely determine and specify the requirements that a software system must satisfy.
2) How to efficiently, accurately, and completely determine and specify the safety properties that a system must maintain.
3) How to validate that the specified requirements and safety properties are the desired ones.
4) How to best partition the implementation of requirements and safety properties among software, hardware, and humans.

[**] Henceforth, we use "future vision" as a shorthand for the visions of aviation expressed in the various documents discussed in Section II.

5) How to efficiently, accurately, and with a sufficiently high degree of confidence verify that a software system satisfies all its requirements and maintains all its safety properties.
6) How to demonstrate to others, such as certification authorities, that all necessary verification and validation has been completed.
7) How to ensure the integrity and accuracy of all the databases on which a software systems depends.
8) When accidents do occur, how to effectively diagnose the software contributions to the accidents, so that future systems will not be susceptible to similar accidents.

Partial solutions exist for some of these. These partial solutions are adequate for many current software systems; however, the complex and automated systems required to realize the future vision will require software that is far more complex and pervasive than exists today. Partial solutions to fundamental problems will not be enough. A research program designed to discover full solutions is needed.

Creating such a program will not be an easy task. It will require unprecedented cooperation among relevant government agencies, industry, and academia. Also, since much of the current research in safety-critical software is taking place outside of the United States, the research program will require international cooperation and coordination. We are by no means the first to recognize the need for such a program;[30] however, if step one is completed as it should be, then perhaps we will be, in some sense, the last.

C. Step Three.

Solving these problems through research alone will not be enough. The solutions must become part of industry practice and embodied in the rules, regulations, and guidelines used to certify systems. These rules and regulations govern the regulatory oversight of the manufacture and equipage of aircraft, of the operation of aircraft within the national airspace, and in the provision of air traffic services. This oversight is commonly referred to as certification, and it will be required for the new technologies proposed in the future vision. Thus, the third necessary step on the way forward is the creation, advocacy, and execution of a robust program to address certification. To be successful, this step must be taken in tandem with step two.

Difficulties in certification are already being encountered even with systems significantly less complex and automated than those required by the future vision. For example, as far back as 1999, an FAA-commissioned study noted the following: "…during the last decade or so, the dynamic growth and globalization of aviation have outpaced the government's certification policies and regulatory oversight of Communications, Navigation, Surveillance/Air Traffic Management (CNS/ATM) systems, equipment and procedures. The time and cost of implementing new operational capabilities is increasing and is inhibiting the introduction of safety enhancements. … This situation has caused many in governments and industry to observe that the current certification process is too lengthy and too costly." [34]

Certification of software-intensive systems is particularly problematic primarily because of the gaps in our understanding of how to develop complex software systems. As a result of these gaps, certification of such systems today is largely based on adherence to specified processes, rather than on a systematic evaluation of the safety and quality of the systems themselves. This process-based approach to certification is likely a primary reason for excessive costs and time.[35] An effective program must have as one of its primary goals the replacement of the process-based approach with an approach (or approaches) based on a systematic evaluation of the systems themselves.

Creating this program will not be an easy task either. National and international cooperation and collaboration will be just as necessary for this program as for the research program.

V. Concluding Remarks

Many visions of the future rely on automated systems to make air travel safer, more secure, more dependable, and available to more people than it is today. This is a desirable future, but it is not a feasible one, *unless* significant advances are made over the current state-of-the-practice in safety-critical software development and certification. In this paper, we have outlined a three-step process for making these advances: (1) recognize that increasing automation is not problem-free; (2) create and execute a robust software research program designed to close the existing gaps in our understanding of how to develop safe, highly-automated software systems; and (3) embody the results of this research program in rules and guidelines for certification of these systems. These steps (or ones like them) may help us achieve that future vision.

Acknowledgments

We thank Professor John Knight and his students (particularly Billy Greenwell and Elisabeth Strunk) for the consistent encouragement and intellectual stimulation that they provide to us.

References

[1]Federal Aviation Administration, "Flight Plan 2005-2009," URL: http://www.faa.gov/about/plans_reports/ [cited 12 September 2005].

[2]Federal Aviation Administration, "Blueprint for NAS Modernization, 2002 Update," URL: http://www.faa.gov/nasarchitecture/BlueprintURLs.htm [cited 12 September 2005].

[3]National Aeronautics and Space Administration, "NASA Aeronautics Blueprint: Towards a Bold New Era in Aviation," URL: http://www.aerospace.nasa.gov/aboutus/tf/aero_blueprint/index.html [cited 12 September 2005].

[4]Office of the Secretary of Defense, "Airspace Integration Plan for Unmanned Aviation," November 2004, URL: http://www.acq.osd.mil/uas/ [cited 12 September 2005].

[5]Joint Planning and Development Office, "Next Generation Air Transportation System: Integrated Plan," December 2004, URL: http://www.jpdo.aero/site_content/NGATS_v1_1204.pdf [cited 21 September 2005].

[6]Holloway, C. Michael and Hayhurst, Kelly J., "Software System Safety & the NASA Aeronautics Blueprint," *Proceedings of the 21st International System Safety Conference*, System Safety Society, Unionville, VA, 2003, pp. 1183-1192.

[7]General Accounting Office, "Aircraft Certification, New FAA Approach Needed to Meet Challenges of Advanced Technology," GAO/RCED-93-155, September 1993.

[8]Aeronautical Civil of the Republic of Columbia, "Aircraft Accident Report: Controlled Flight into Terrain, America Airlines Flight 965, Boeing 757-223, N651AA, Near Cali, Columbia, December 20, 1995," Santafe De Bogotoa, D.C., Columbia, 1996. Web version prepared by Peter Ladkin. URL: http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Cali/calierep.html [cited 14 September 2005].

[9]Main Commission Aircraft Accident Investigation Warsaw, "Report on the Accident to Airbus A320-211 Aircraft in Warsaw on 14 September 1993," Warsaw, Poland, March 1994. Web version prepared by Peter Ladkin, URL: http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Warsaw/warsaw-report.html [cited 15 September 2005].

[10]National Transportation Safety Board, "Flight into Terrain during Missed Approach USAir 1016, DC-9-31, N954VJ Charlotte/Douglas International Airport Charlotte, North Carolina July 2, 1994," NTSB Report Number: AAR-95-03, adopted on April 4, 1995.

[11]Dornheim, Michael A., "Codes Gone Awry," *Aviation Week & Space Technology*, February 28, 2005, p. 63.

[12]General Accounting Office, "National Airspace System, Status of FAA's Standard Terminal Automation Replacement System," GAO-02-1071, September 2002.

[13]Office of Inspector General, "Action: Follow-Up on Federal Aviation Administration's Acquisition of Standard Terminal Automation Replacement System", memorandum to the FAA dated 6/3/2002, Control No. 2002-087.

[14]The Associated Press, "Controversy Over Air Traffic System," posted on CBSNews.com, June 5, 2002. URL: http://www.cbsnews.com/stories/2002/06/05/national/printable511149.shtml [cited 20 September 2005].

[15]Wisely, John, "Glitches Plague Airport Radar," *The Detroit News*, posted on detnews.com, April 9, 2004. URL: http://www.detnews.com/2004/business/0404/11/a01-118041.htm [cited 20 September 2005].

[16]Office of Inspector General, "Status of FAA's Major Acquisitions: Cost Growth and Schedule Delays Continue to Stall Air Traffic Modernization," Report Number AV-2005-061, May 26, 2005.

[17]National Aeronautics and Space Administration, "Mars Climate Orbiter Mishap Investigation Board Phase I Report," November 10, 1999.

[18]Jet Propulsion Laboratory, JPL Special Review Board, "Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions," JPL D-18709, March 22, 2000.

[19]Johnson, C. W., "The Natural History of Bugs: Using Formal Methods to Analyse Software Related Failures in Space Missions," edited by J.S. Fitzgerald, I.J. Hayes, and A. Tarlecki, *Lecture Notes in Computing Science Number 3582*, Heidelberg, Germany, 2005, pp. 9-25.

[20]Government Accountability Office, "Tactical Aircraft, F/A-22 and JSF Acquisition Plans and Implications for Tactical Aircraft Modernization, Statement of Michael Sullivan, Director, Acquisition and Sourcing Management Issues," GAO-05-519T, April 6, 2005.

[21]Wall, Robert, "Code Red Emergency," *Aviation Week & Space Technology*, June 9, 2003, pp. 35-36.

[22]General Accounting Office, "Tactical Aircraft, Status of the F/A-22 Program, Statement of Allen Li, Director, Acquisition and Sourcing Management" GAO-33-603T, April 2, 2003.

[23]Selinger, Marc, "F/A-22's Software Stability 'No Longer An Issue,' USAF Says," *Aerospace Daily*, May 3, 2004, URL: http://www.aviationnow.com/avnow/news/channel_aerospacedaily_story.jsp?id=news/vic05034.xml [cited 21 September 2005]

[24]U.S. Air Force, AIRCRAFT ACCIDENT INVESTIGATION, F/A-22 S/N 00-4014, URL: http://www.airforcetimes.com/content/editorial/pdf/af.exsum_f22crash_060805.pdf [cited 14 September 2005].

[25]Trimble, Stephen, "Defect Downed F/A-22," *Flight International*, June 14, 2005, URL: http://www.flightinternational.com/Articles/2005/06/14/Navigation/194/199029/+Defect+downed+FA-22.html [cited 21 September 2005].

[26]Trimble, Stephen, "Avionics Redesign Aims to Improve F/A-22 Stability," *Flight International*, August 23, 2005, URL: http://www.flightinternational.co.uk/Articles/2005/08/23/Navigation/181/201118/Avionics+redesign+aims+to+improve+FA-22+stability.html [cited 21 September 2005].

[27]Ratnam, Gopal, "Paying the Price: Technology Paces Weapon Costs," *DefenseNews.com*, March 7, 2005, URL: http://tinyurl.com/cc4ll [cited 15 September 2005].

[28]General Accounting Office, "National Airspace System, Persistent Problems in FAA's New Navigation System Highlight Need for Periodic Reevaluation," GAO/RCED/AIMD-00-130, June 2000.

[29]Government Accountability Office, "National Airspace System, FAA Has Made Progress but Continues to Face Challenges in Acquiring Major Air Traffic Control Systems," GAO-05-331, June 2005.

[30]Center for National Software Studies, "Software 2015: A National Software Strategy to Ensure U.S. Security and Competitiveness", April 29, 2005.

[31]Charette, Robert N., "Why Software Fails," *IEEE Spectrum*, September 2005, pp. 42-49.

[32]Brooks, Fredrick P., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 20, No. 4, April 1987, pp. 10-19.

[33]Dijkstra, Edsger W., "The End of Computing Science?" *Communications of the ACM,* Vol. 44, No. 3, March 2001, p. 92.

[34]RTCA Task Force 4, "Final Report of RTCA Task Force 4, Certification," RTCA, Inc., Washington, D.C., February 26, 1999.

[35]Hayhurst, Kelly J., Dorsey, Cheryl A., Knight, John C., Leveson, Nancy G., and McCormick, G. Frank, "Streamlining Software Aspects of Certification: Report on the SSAC Survey," NASA/TM-199-209519, August 1999.